```
   SSSSSSSSSSS  DDDDDDDDDDD        AAAAAAAAA
   SSSSSSSSSSS  DDDDDDDDDDDD       AAAAAAAAA
   SSSSSSSSSSS  DDDDDDDDDDDD       AAAAAAAAA
SSS             DDD          DDD   AAA         AAA
SSS             DDD          DDD   AAA         AAA
SSS             DDD          DDD   AAA         AAA
SSS             DDD          DDD   AAA         AAA
SSS             DDD          DDD   AAA         AAA
SSS             DDD          DDD   AAA         AAA
   SSSSSSSSS    DDD          DDD   AAA         AAA
   SSSSSSSSS    DDD          DDD   AAA         AAA
   SSSSSSSSS    DDD          DDD   AAA         AAA
            SSS DDD          DDD   AAAAAAAAAAAAAAA
            SSS DDD          DDD   AAAAAAAAAAAAAAA
            SSS DDD          DDD   AAAAAAAAAAAAAAA
            SSS DDD          DDD   AAA         AAA
            SSS DDD          DDD   AAA         AAA
            SSS DDD          DDD   AAA         AAA
SSSSSSSSSSS     DDDDDDDDDDDD       AAA         AAA
SSSSSSSSSSS     DDDDDDDDDDDD       AAA         AAA
SSSSSSSSSSS     DDDDDDDDDDDD       AAA         AAA
```

FILEID**VAXOPS

```
VV      VV    AAAAAA    XX      XX    000000    PPPPPPPP      SSSSSSSS
VV      VV    AAAAAA    XX      XX    000000    PPPPPPPP      SSSSSSSS
VV      VV   AA    AA   XX      XX   00    00   PP     PP    SS
VV      VV   AA    AA   XX      XX   00    00   PP     PP    SS
VV      VV   AA    AA     XX  XX     00    00   PP     PP    SS
VV      VV   AA    AA     XX  XX     00    00   PP     PP    SS
VV      VV   AA    AA       XX       00    00   PPPPPPPP      SSSSSS
VV      VV   AA    AA       XX       00    00   PPPPPPPP      SSSSSS
VV      VV   AAAAAAAAAA     XX  XX   00    00   PP                  SS
VV      VV   AAAAAAAAAA     XX  XX   00    00   PP                  SS
  VV  VV     AA    AA   XX      XX   00    00   PP                  SS      ....
  VV  VV     AA    AA   XX      XX   00    00   PP                  SS      ....
    VV       AA    AA   XX      XX     000000   PP          SSSSSSSS        ....
    VV       AA    AA   XX      XX     000000   PP          SSSSSSSS        ....

RRRRRRRR    EEEEEEEEEE     QQQQQQ
RRRRRRRR    EEEEEEEEEE     QQQQQQ
RR      RR  EE            QQ      QQ
RR      RR  EE            QQ      QQ
RR      RR  EE            QQ      QQ
RR      RR  EE            QQ      QQ
RRRRRRRR    EEEEEEEE      QQ      QQ
RRRRRRRR    EEEEEEEE      QQ      QQ
RR  RR      EE            QQ  QQ  QQ
RR  RR      EE            QQ  QQ  QQ
RR    RR    EE            QQ    QQ
RR    RR    EE            QQ    QQ
RR      RR  EEEEEEEEEE      QQQQ  QQ
RR      RR  EEEEEEEEEE      QQQQ  QQ
```

```
!  VAXOPS.REQ - OP CODE TABLE FOR VAX INSTRUCTIONS
! Version:      'V04-000'
```



```
!  Author:
!               KEVIN PAMMETT, MARCH 2, 1977.
!
! Modified by:
!
!       V001    TMH0001         Tim Halvorsen   09-Feb-1981
!               Rewrite macro invocations to supply the entire SRM
!               operand specification, to allow checking for literals
!               in write operands, and other invalid conditions.
!--

LITERAL

!
! OPERAND ACCESS TYPE (A,B,M,R,V,W) - 1 BIT WIDE

        ACCESS_A = 0,                   ! EFFECTIVE ADDRESS
        ACCESS_B = 0,                   ! BRANCH DISPLACEMENT
        ACCESS_R = 1,                   ! OPERAND IS READ-ONLY
        ACCESS_W = 0,                   ! OPERAND IS WRITE-ONLY
        ACCESS_M = 0,                   ! OPERAND IS MODIFIED
        ACCESS_V = 0,                   ! ADDRESS A SET OF 2 REGISTERS

!
! OPERAND DATA TYPE (B,W,L,Q,F,D,G,H,V) - 3 BITS WIDE

        DATA_B = 0,                     ! BYTE CONTEXT
```

```
        DATA_W = 1,                     ! WORD CONTEXT
        DATA_L = 2,                     ! LONGWORD CONTEXT
        DATA_Q = 3,                     ! QUADWORD CONTEXT
        DATA_F = DATA_L,                ! FLOATING CONTEXT
        DATA_D = DATA_Q,                ! FLOATING DOUBLE CONTEXT (8 BYTES)
        DATA_G = DATA_Q,                ! FLOATING GRAND CONTEXT (8 BYTES)
        DATA_H = 4,                     ! FLOATING HUGE CONTEXT (16 BYTES)

!
! BRANCH DISPLACEMENT TYPES
!

        NO_BRANCH = 0,                  ! NO BRANCH
        BRANCH_BYTE = 1,                ! BRANCH BYTE
        BRANCH_WORD = 2;                ! BRANCH WORD

!
! THE FOLLOWING MACRO IS USED TO BUILD SUCCESSIVE ENTRIES FOR
! THE TABLE.  EACH MACRO CALL CONTAINS THE
! INFO FOR 1 VAX OPCODE, AND THE ENTRIES ARE SIMPLY
! BUILT IN THE ORDER THAT THE MACRO CALLS ARE MADE -
! THE ASSUMPTION IS THAT THEY WILL BE MADE IN ORDER OF
! INCREASING OPCODE VALUES.  THIS IS NECESSARY BECAUSE
! THE TABLE IS ACCESSED BY USING A GIVEN OPCODE AS THE
! TABLE INDEX.
!

COMPILETIME $BRANCH_TYPE=0;

MACRO
    GET_1ST(A,B) = A%,
    GET_2ND(A,B) = B%,
    OPERAND(NAME) =
        %IF %NULL(NAME)
        %THEN
            0
        %ELSE
            BEGIN
            %IF NOT %DECLARED(%STRING('ACCESS_',GET_1ST(%EXPLODE(NAME))))
            %THEN
                %WARN('Invalid access type ',GET_1ST(%EXPLODE(NAME)))
            %FI
            %IF NOT %DECLARED(%STRING('DATA_',GET_2ND(%EXPLODE(NAME))))
            %THEN
                %WARN('Invalid data type ',GET_2ND(%EXPLODE(NAME)))
            %FI
            %IF NAME EQL 'BB'
            %THEN
                %ASSIGN($BRANCH_TYPE, BRANCH_BYTE)
            %ELSE %IF NAME EQL 'BW'
            %THEN
                %ASSIGN($BRANCH_TYPE, BRANCH_WORD)
            %FI %FI
            %NAME('DATA_',GET_2ND(%EXPLODE(NAME))) +
                %NAME('ACCESS_',GET_1ST(%EXPLODE(NAME))) ^ 3
            END
```

```
        %FI %,

    OPDEF(NAME, OPC, OP1, OP2, OP3, OP4, OP5, OP6) =
        %ASSIGN($BRANCH_TYPE,NO_BRANCH)
        %RAD50_11 NAME,                         ! Opcode name in RAD50
        %IF GET_1ST(%EXPLODE(NAME)) EQL 'X'     ! If undefined opcode,
            AND GET_2ND(%EXPLODE(NAME)) EQL 'X'
        %THEN
            NOT_AN_OP                           ! then no operands
        %ELSE
            %LENGTH-2                           ! else, number of operands
        %FI OR
            OPERAND(OP1)^4,                     ! Define each operand
        OPERAND(OP2) OR
            OPERAND(OP3)^4,
        OPERAND(OP4) OR
            OPERAND(OP5)^4,
        OPERAND(OP6) OR
            $BRANCH_TYPE^4%;                    ! Define branch context

!
! MACROS TO ACCESS THE FIELDS.
!

MACRO
    OP_NAME      = 0,0,32,0%,                   ! OPCODE MNEUMONIC (6 RAD50 CHARS)
    OP_NUMOPS    = 4,0,4,0%,                    ! NUMBER OF OPERANDS
    OP_CONTEXT(I) = 4+I/2, ((I) AND 1)*4, 3, 0 %, ! OPERAND CONTEXT
    OP_DATATYPE(I) = 4+I/2, ((I) AND 1)*4 + 3, 1, 0 %, ! OPERAND DATA TYPE
    OP_BR_TYPE = 7,4,4,0 %;                     ! CONTEXT OF BRANCH DISPLACEMENT

LITERAL
        OPTSIZE = 8,                    ! EACH OPINFO BLOCK IS 9 BYTES LONG.
        MAXOPCODE = %X'FD',             ! MAXIMUM VAX OP CODE WHICH IS VALID.
        MAXOPRNDS = 6,                  ! MAXIMUM NUMBER OF OPERANDS PER INSTRUCTION.
                                        ! NO INSTRUCTION THAT HAS BRANCH TYPE ADDRESSING
                                        ! CAN HAVE THIS MANY OPERANDS UNLESS WE CHANGE
                                        ! THE ORGANIZATION OF EACH OPINFO BLOCK.
        BITS_PER_BYTE = 8,              ! NUMBER OF BITS IN A VAX BYTE.
        AP_REG  = 12,                   ! NUMBER OF PROCESSOR REGISTER, 'AP'.
        PC_REG  = 15,                   ! NUMBER OF PROCESSOR REGISTER, 'PC'.

        PC_REL_MODE = 8,                ! ADDRESSING MODE: (PC)+
        AT_PC_REL_MODE = 9,             ! ADDRESSING MODE: @(PC)+
        INDEXING_MODE = 4,              ! ADDRESSING MODE: XXX[RX]

        SHORT_LIT_AMODE = 0,            ! Short literals fit right into the mode byte.
        REGISTER_AMODE  = 5,            ! Register mode addressing.
        REG_DEF_AMODE   = 6,            ! Register deferred addressing mode.
        AUTO_DEC_AMODE  = 7,            ! Auto decrement addressing mode.
        AUTO_INC_AMODE  = 8,            ! Auto Increment addressing mode.
        DISP_BYTE_AMODE = 10,           ! All of the displacement modes start from
                                        !  here.  See ENC_OPERAND() IN DBGENC.B32
        DISP_LONG_AMODE = 14,
        OP_CH_SIZE = 6;                 ! SIZE, IN ASCII CHARS, OF OPCODE MNEMONIC.
```

```
MACRO
        DSPL_MODE = 0,4,4,0 %,  ! ADDRESSING MODE BITS FROM THE DOMINANT MODE
                                !  BYTE OF AN OPERAND REFERENCE.
        DOM_MOD_FIELD = 0,5,2,1 %,
                                ! BITS WHICH WE PICK UP TO DIFFERENTIATE CERTAIN
                                !  TYPES OF DOMINANT MODES.  SEE DBGMAC.B32
        SHORT_LITERAL = 0,0,6,0 %,
                                ! HOW TO EXCTRACT A 'SHORT LITERAL' FROM
                                !   THE INSTRUCTION STREAM.  SEE SRM.
        AMODE   = 0,4,4,1 %,    ! BITS OF DOMINANT MODE ADDRESSING BYTE
                                !   WHICH SPECIFY THE ACTUAL MODE.
        AREG    = 0,0,4,0 %,    ! BITS OF DOMINANT MODE ADDRESSING BYTE
                                !   WHICH SPECIFY REGISTER NUMBER, ETC.
        NOT_AN_OP = 15 %,       ! OP_NUMOPS INDICATOR FOR UNASSIGNED OPCODES.
        RESERVED = 'UNUSED' %;  ! NAME OF RESERVED OPCODES.

MACRO
        NEXT_FIELD(INDEX)       ! USED TO GET THE ADDRESS OF THE NEXT
                                !   FIELD OF A BLOCK.
                = (INDEX),0,0,0 %;

!  MACROS AND LITERALS SPECIFICALLY FOR INSTRUCTON ENCODING.
!  ('MACHINE -IN'.)

LITERAL
        BAD_OPCODE      = 1,    ! CAN'T INTERPRET THE GIVEN ASCII OPCODE.
        BAD_OPERAND     = 2,    ! UNDECODABLE OPERAND REFERENCE.
        BAD_OPRNDS      = 3,    ! WRONG NUMBER OF OPERANDS.
        INS_RESERVED    = 4;    ! GIVEN OPCODE IS RESERVED.

LITERAL
                ! We only have to special-case a few OPCODES,

        OP_CASEB        = %X'8F',
        OP_CASEW        = %X'AF',
        OP_CASEL        = %X'CF';
!++
!
!       TOKEN VALUES USED FOR ENCODING/DECODING
!
!--

LITERAL
        indexing_token  = 240,
        val_token       = 241,
        byte_val_token  = val_token + %SIZE(VECTOR[1,BYTE]),    ! 242
        word_val_token  = val_token + %SIZE(VECTOR[1,WORD]),    ! 243
        brch_token      = 244,
        long_val_token  = val_token + %SIZE(VECTOR[1,LONG]),    ! 245
        at_reg_token    = 246,
        register_token  = 247,
        lit_token       = 248,
        bad_token       = 249;
```

```
! The following structure declaration selects the proper opcode
! table by looking for the extended opcode opcode(s).

STRUCTURE OPCODE_TBL [OPC,O,P,S,E] =
        BEGIN
        EXTERNAL LIB$GB_OPINFO1 : BLOCKVECTOR[256,OPTSIZE,BYTE];
        EXTERNAL LIB$GB_OPINFO2 : BLOCKVECTOR[256,OPTSIZE,BYTE];
        LOCAL OFFSET;
        OFFSET = 0;
        IF (OPC AND %X'FF') NEQ %X'FD'
        THEN LIB$GB_OPINFO1[OPC,,OFFSET,0,8,0]  ! One byte opcodes
        ELSE LIB$GB_OPINFO2[(OPC^-8),,OFFSET,0,8,0]     ! Two byte opcodes
        END<P,S,E>;

!       VAXOPS.REQ      - last line
```

XCASE
LIS

SDADEF
SDL

RMSDEF
MAR

UNPUS
LIS

SDATEST
MAP

SDA

SDA
MAP

DCLDEF
MAR

SCSDEF
MAR

TPR
LIS

XTAB
LIS

IPROBE
LIS

SYSDEF
MAR

IMGDEF
MAR

NETDEF
MAR

VAXOPS
REQ

MACROS
MAR

CLUSTER
LIS